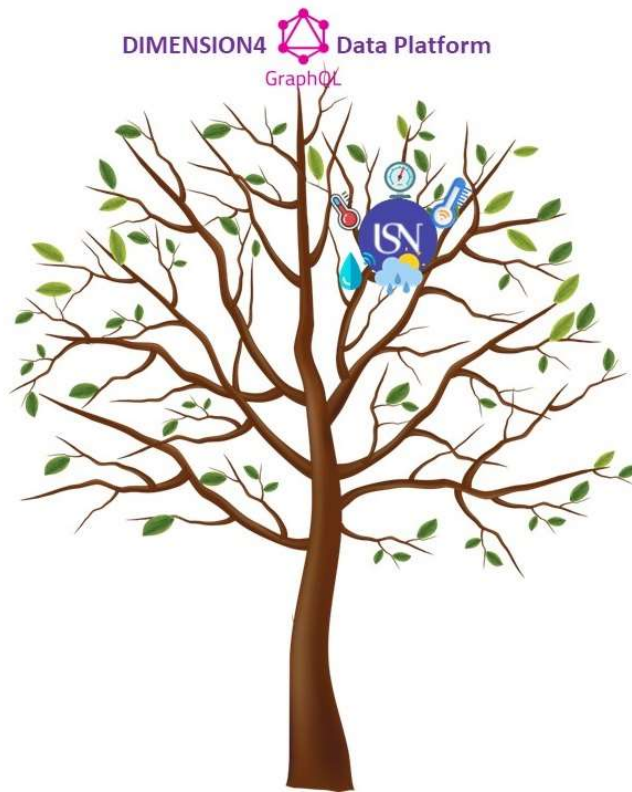


FM4017 Project 2022

Development of Weather System with Interface to IoT GraphQL Data Platform



Group code: MP-16-22

Faculty of Technology, Natural sciences and Maritime Sciences
Campus Porsgrunn

Course: FM4017 Project, 2022

Title: Development of Weather System with Interface to IoT GraphQL Data Platform

This report forms part of the basis for assessing the students' performance on the course.

Project group: MP-16-22

Group participants: Hamed Jalalian Javadpour
Shirin Shahrokh
Martin Dizbite Ose

Supervisor: Hans-Petter Halvorsen

Project partner: Dimension Four

Summary:

DimensionFour is a scalable end-to-end IoT service, that provides cloud storage for sensors or other data sources through a powerful GraphQL API. The goal of this project has been to build a showcase solution utilizing a MicroSun weather station located at USN Porsgrunn as a data source, and DimensionFour as a data storage platform.

This has been done by creating a .NET-based application for connecting to the weather station through the intranet and exporting the data to DimensionFour's service and developing a web application using modern web technologies to retrieve and visualize the stored data.

Preface

The "Development of Weather System with Interface to IoT GraphQL Data Platform" project, has tried to retrieve data from the Weather System" at the University of South-Eastern Norway (USN), in Porsgrunn and sent the data to "Dimension Four". The software has two main parts, retrieving data and sending it to Dimension Four as the first part, and the second part, monitoring the retrieved data and showing them as a website.

The members of this project would like to appreciate USN, the supervisor, Hans-Petter Halvorsen, the external partner "Dimension Four", and all other staff that helped them during this project.

It is worth mentioning, in this project, due to the geographical distance between the members, it was good practice for teamwork and effective communication although it was not easy.

Porsgrunn, 18/11/2022

Hamed Jalalian Javadpour
Shirin Shahrokh
Martin Dizbite Ose

Contents

- 1 Introduction7
 - 1.1 Project Overview7
 - 1.2 Overview of existing Weather Station8
 - 1.3 Previous student work10
 - 1.4 DimensionFour Platform11
 - 1.5 DimensionFour & Peer technology11
 - 1.6 GraphQL12
- 2 Methods15
 - 2.1 Design of Datalogging Software15
 - 2.1.1 UML Design Steps15
 - 2.1.2 Implement and Testing18
 - 2.2 Development of Data Monitoring Application24
 - 2.2.1 Web Technologies used for the Web Application24
- 3 Result26
- 4 Discussion and Conclusion28

Table of figures

<i>Figure 1-1: The system sketch.....</i>	<i>7</i>
<i>Figure 1-2: Home page of Weather System</i>	<i>8</i>
<i>Figure 1-3: Realtime Display page on Weather System</i>	<i>9</i>
<i>Figure 1-4: Latest Measurements on Weather System</i>	<i>9</i>
<i>Figure 1-5: a sample of a result in a JSON format</i>	<i>13</i>
<i>Figure 2-1: The class diagram of datalogging software</i>	<i>16</i>
<i>Figure 2-2: Domain Model of the datalogging software.....</i>	<i>16</i>
<i>Figure 2-3: The first class diagram of the datalogging software</i>	<i>17</i>
<i>Figure 2-4: Logging Form</i>	<i>18</i>
<i>Figure 2-5: Main Page Form</i>	<i>18</i>
<i>Figure 2-6: Parameters information Form.....</i>	<i>19</i>
<i>Figure 2-7: Chosen parameters Form</i>	<i>19</i>
<i>Figure 2-8: The sample of retrieved data from weather System</i>	<i>20</i>
<i>Figure 2-9: Technologies are used for developing the data monitoring web application.</i>	<i>24</i>
<i>Figure 3-1: Data monitoring web application main screen.</i>	<i>26</i>
<i>Figure 3-2: Full size of temperature chart with mode details.</i>	<i>26</i>
<i>Figure 3-3: Historical data page.</i>	<i>27</i>
<i>Figure 3-4: Error message when the API URL is misconfigured or the cloud server is down.</i>	<i>27</i>

Nomenclature

XML	Extensible Markup Language
REST	Representational state transfer
API	Application Programming Interface
LabVIEW	Laboratory Virtual Instrumentation Engineering Workbench
HTTP	The Hypertext Transfer Protocol
CSV	Comma-Separated Values
GUI	Graphical User Interface
SQL	Structured Query Language
hPa	Hectopascal

1 Introduction

The "Development of Weather System with Interface to IoT GraphQL Data Platform" project, has tried to retrieve data from the Weather System" at the University of South-Eastern Norway (USN), in Porsgrunn and sent the data to "Dimension Four". The software has two main parts, retrieving data and sending it to Dimension Four. The data logging software is developed in C# - Visual Studio software as the first part and monitoring the retrieved data and showing them as a web page as the second part. The data monitoring application is developed in React JS library using JavaScript, CSS, HTML and it is deployed on Netlify cloud service. Some additional feature has been added for the security of the software to control the user login. For this part of the software, SQL Server has been used.

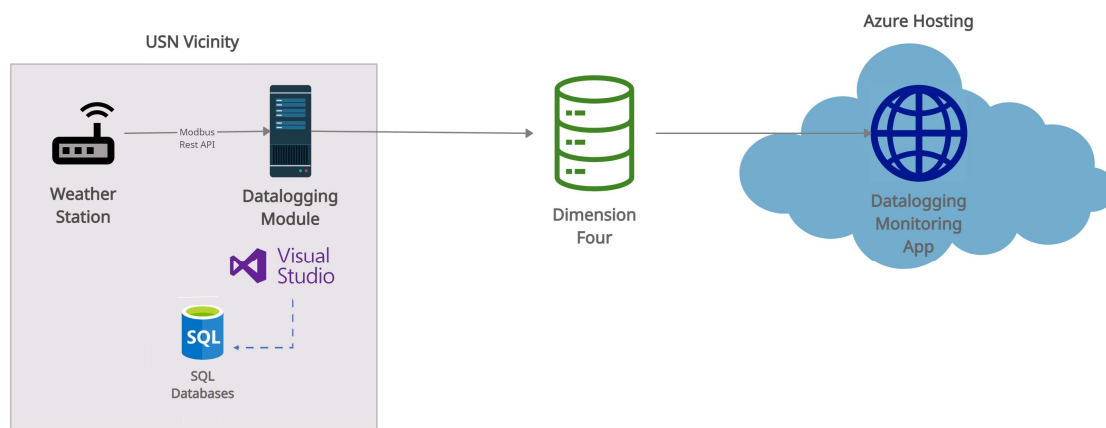


Figure 1-1 The system sketch

1.1 Project Overview

The main goal of the project work is to utilize DimensionFour as an IoT cloud platform to store and retrieve weather data from. The Weather Station is located at USN Porsgrunn and is only connected to the university intranet. A separate application for retrieving data and exporting to DimensionFour was therefore necessary, and another separate web application for displaying the data. The work was therefore naturally divided into developing a Datalogging Application and a Data Monitoring Application. GitHub was used as code repository for the Datalogging Application, since two people were working on that part. Microsoft Teams was used as a means for communication, as several of the group members live outside of Porsgrunn. Weekly status meetings were held with the project supervisor, and a kanban-board in Teams was used to track progress of the project tasks.

1.2 Overview of existing Weather Station

The Weather Station at USN is a Weather MicroServer from Columbia Weather Systems and it is located in a cabinet near the fire house on second floor of the C-building, also it is accessible from a Static IP-address at USN (Room A194 and A195) by using a username and password.

After getting access to the Weather System by a browser, different parts can be seen as follows:

- Home page (Figure 1-2)
- Network Setup (Figure 1-3)
- Change Password (Figure 1-4)
- Date and Time
- Data Logs
- Data Outputs
- Configuration File
- Measurements
- Units
- Parameter Settings
- Update Firmware
- Diagnostics

A view of the first three sections is as follows in Figure 1-2:

Home page:

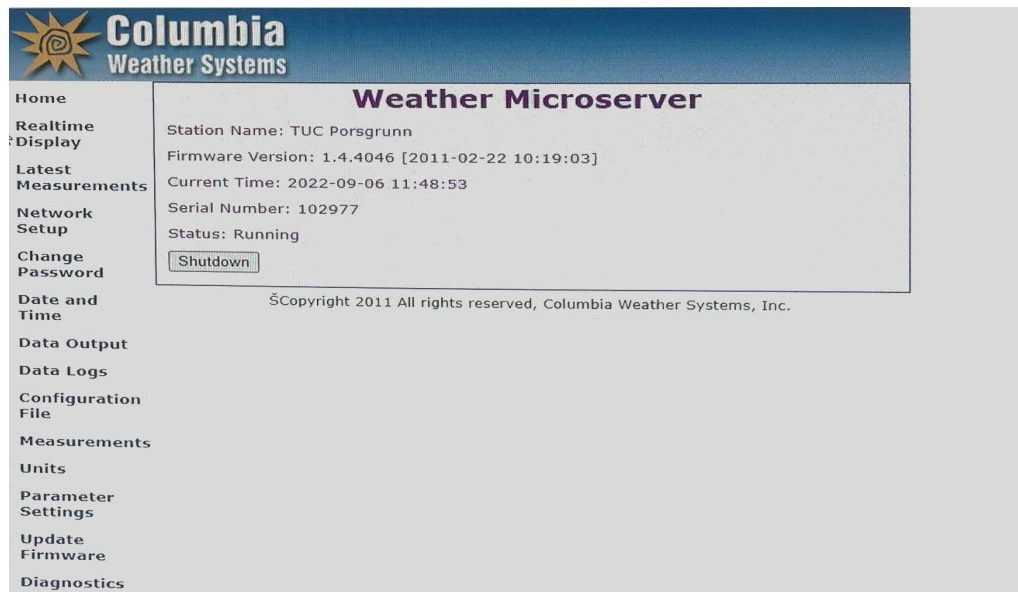


Figure 1-2: Home page of Weather System

- Realtime Display page: Figure1.3:

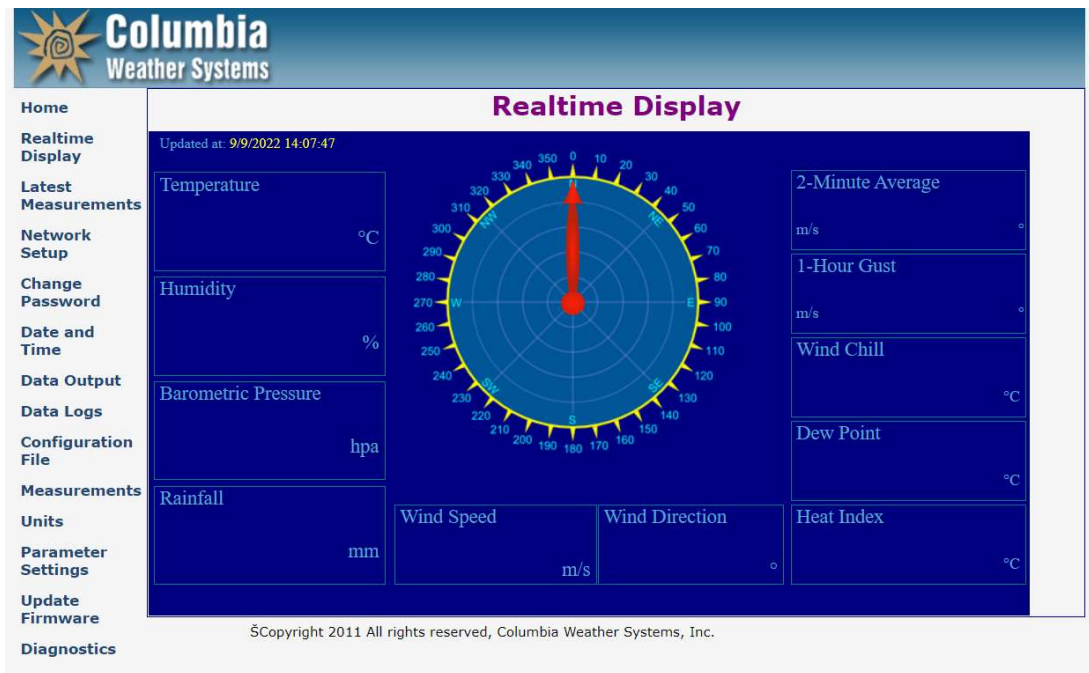


Figure 1-3: Realtime Display page on Weather System

- Latest Measurements page: 1.4:

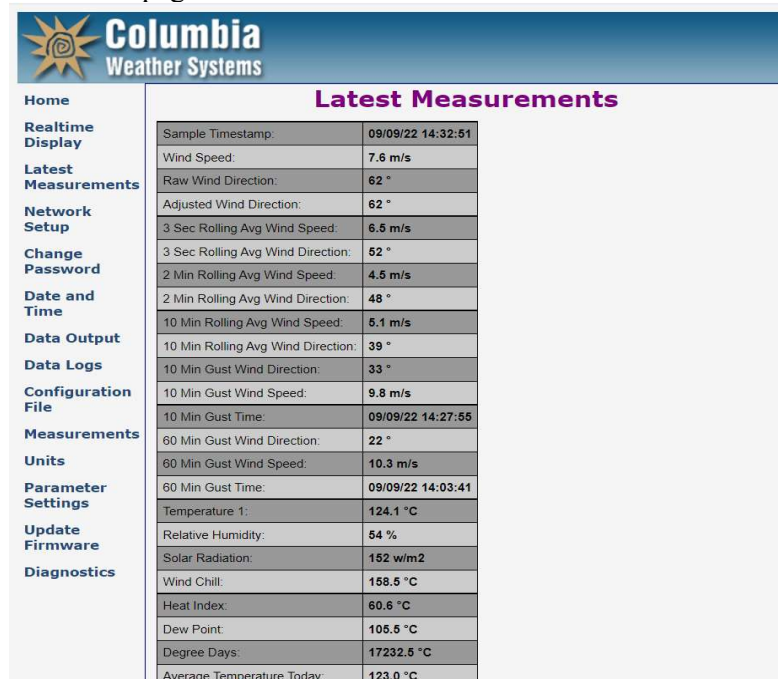


Figure 1-4: Latest Measurements on Weather System

The weather station supplies interfaces to ModBus, XML web service, an online portal (as shown above), and an FTP output. The output of the FTP server includes both XML and CSV. The XML output is also accessible through an XML web service. A sample excerpt of the “current data” endpoint output can be seen below:

```
<oriondata station='TUC'>

  <meas name="mtSampTime">2022/09/22 12:02:58</meas>

  <meas name="mtWindSpeed">0.0</meas>

  <meas name="mtRawWindDir">141</meas>

  ...

  <meas name="mtRainThisYear">3.82</meas>

  <meas name="mtRainRate">0.00</meas>

  <meas name="mtHailToday">0</meas>

  <meas name="mtHailRate">0</meas>

</oriondata>
```

After retrieving values from the weather station, one might have to convert some of the measurements, as the weather station might be configured to imperial units.

1.3 Previous student work

There already exists two master thesis papers from USN that have utilized the weather station. One of them is “Design and Implementation of Weather System for Acquiring and Monitoring of Weather Data” [2]. In this paper, the students aimed at designing and implementing a weather system, utilizing LabVIEW’s G-language through ModBus protocol, and storing data in a local SQL database. Furthermore, OPC connectivity was established, so that users in the vicinity of the USN premises would be able to retrieve data in real-time. Finally, an ASP.NET website was made to display the weather data, as well as a web service that made the data accessible through internet.

The paper resulted in an implemented website with the back-end system necessary to retrieve data from the weather station and display it to users. It was accessible through the address <http://128.39.35.252/Weather/> - presumably hosted locally on the university campus. There were various visualizations and widgets on the site to show the weather data, which was refreshed once a minute.

Another paper written in 2013, which also utilized the weather system, focused on creating a Windows Store App that would be able to present data from the weather station. At the time, Windows Store was a new feature that came with Windows 8. The technical implementation of the app used C# and XAML to create backend logic and a GUI.

1.4 DimensionFour Platform

Developing a highly scalable and maintainable Internet of Things (IoT) service is a challenge. This could be included frontend to handle user interface and backend with an integrated database to take responsibility of storing data and handling requests. With modern and fast Dimension4 GraphQL API cloud server, focus on project requirement and let Dimension4 to handle processing sensor data and payment management system via a simple, secure, and fast GraphQL cloud service. Your data will store on strong GraphQL branches like a tree allowing to select specific piece of data without having data fetching problem even in low storage devices such as mobile phones and tablets.

They offer two standard pricing packages, 'Earth' and 'Star', both of which feature unlimited points/ devices and users. The former comes with 100,000 API calls and 100 MB storage, whereas the latter includes 1,000,000 API calls and 1GB. 'Star' offers SLA, whereas 'Earth' does not. In addition to these two services, Dimension Four also offer a 'Galaxy' package, which is entirely customizable. Dimension Four are also wholly technology agnostic, allowing companies to use their preferred carrier and hardware with their IoT cloud database platform.

1.5 DimensionFour & Peer technology

Internet of Things (or IoT) platforms are essential to any company working with sensor technology, networking, embedded systems, or analytics. They provide a range of services for those looking to analyze and make sense of their data, as well as convenient data storage specialized for IoT purposes. There are, however, many existing IoT platforms out there that can be considered as peer solutions to DimensionFour:

Google Cloud IoT Core

Google Cloud offers their own suite of partner-led IoT solutions, that are hosted on their cloud platform. Among these are ClearObject, Quantiphi, SOTEC and SoftServe. These are implementation partners of Google Cloud, that may assist clients in implementing IoT solutions based on Google Cloud. Furthermore, Aeris, ClearBlade, Dianomic, Leverage, Litmus act as Technology Partners. They offer their IoT platforms and solutions on Google Cloud, and thus as a whole, provides several options of customization for an IoT solution. Their pricing is based on a pay-as-you-go format, based on monthly usage.

Particle

Particle provides an all-in-one solution for hardware, software, and connectivity in regards to IoT solutions. It is therefore a full-stack PaaS (Platform-as-a-Solution) that connects all endpoints from the sensor level, to storage, and out to several applications. They provide their own APIs, SDKs and documentation to allow developers customization of their solutions.

Cisco

Cisco offers both its UCS (Unified Computing System) and Cisco Edge Intelligence as relevant IoT solutions. UCS is one of many cloud platforms that can power a private cloud, which provides the same features. But they are made to be more flexible and provide additional management and safety features for commercial enterprises. Cisco provides APIs and language support for a wide range of programming languages, making integrating Cisco products and services into your existing applications and systems relatively easy. They offer a variety of connectivity options, including private, public, and hybrid clouds. Cisco Edge is a cloud focused ETL (Extract-Transform-Load) tool, that allows enterprises to orchestrate data flow across several cloud platforms. This is very powerful, in the sense that enterprises could implement new data pipelines from new sensors and devices, while orchestrating the data flow together with existing legacy units. Effectively, it makes it easier to perform a digital transformation, as all the orchestration is done through one service. Pricing for Cisco's cloud products are also based on a monthly subscription.[13]

Thingspeak

ThingSpeak offers a hosted, production-ready solution for small to medium-sized IoT installations. This platform's flexibility lies in its ability to accommodate devices from any manufacturer, as its IP-based architecture makes it hardware-neutral. From setting up a connection between your device and the platform to doing any necessary visualization or analysis, you can do it all in one place. Also included are mechanisms for storing real-time data so that engineers can access it across platforms and devices. Thingspeak offers great connectivity, as it allows for protocols such as HTTP, MQTT. It also integrates with Matlab, Arduino, Raspberry Pi, Particle devices and many more with ease. [12]

1.6 GraphQL

DimensionFour is a platform focusing on developer friendly APIs, and thus offer a GraphQL api that is flexible and allows developers to be hardware agnostic. The main idea is to allow developers to use any sensor, without limitations.

GraphQL is a unique query language designed to act as server-side runtime for APIs. Hence, giving clients the exact information they requested. It does this by simplifying data aggregated over multiple APIs. And it later presents only the relevant parts to the client in a single API call. GraphQL aims to make APIs quick and efficient, with relative ease of development [5]. Its vast flexibility enables it to be deployed even in an Integrated Development Environment (IDE). A GraphQL service structures out in the form of types and relevant fields defined by the developer, who later adds a unique function to said fields. The primary operations performed here in GraphQL are [6]:

1. **Queries** – Fetches data from a server.

An example of a query written in GraphQL could be as follows:

```
# GraphQL query with the query and name keywords
query GetPets {
  pets {
    name
    petType
  }
}
```

The query example above gives us the result generally in a JSON format. A typical example of this could be (Figure 1-5):



```
{
  1  "data": {
  2    "pets": [
  3      {
  4        "name": "Sandy",
  5        "petType": "Cat"
  6      },
  7      {
  8        "name": "Hank",
  9        "petType": "Dog"
 10      }
 11    ]
 12  }
}
```

Figure 1-5: a sample of a result in a JSON format

2. **Mutations** – Creates and changes data present on the server end.

An example of a mutation in GraphQL could look like this [7]:

```
mutation CreateReviewForEpisode($ep: Episode!, $review: ReviewInput!) {
  createReview(episode: $ep, review: $review) {
    stars
    commentary
  }
}
```

Where the variables defined would be as follows:

```

mutation CreateReviewForEpisode($ep: Episode!, $review: ReviewInput!) {
  createReview(episode: $ep, review: $review) {
    stars
    commentary
  }
}

```

What we have mentioned so far barely scratches the surface. There are tons of other features loaded within GraphQL that you can add to your query to simplify it. These include Arguments, Aliases, Fragments, Operation names, Variables, Directives, Mutations, Inline Fragments, etc. And apart from queries and mutations, a GraphQL server assorts all the available data in an orderly manner through schemas. A schema is a factor that determines the shape of the info you present by producing a hierarchy of the types with fields of data stored at your back end. The schema also defines available queries and mutations for a client to execute. The schema is primarily built at the heart of any GraphQL Server Implementation with the help of any programming language. The interface itself can be built around it [8].

Peer Technologies – GraphQL vs REST

The main competition faced by GraphQL is REST, Redis, Neo4j, Aerospike, Azure Cosmos DB, etc. And although most of these are unique in nature, they necessarily provide the same function [9]. However, this is not the case for REST.

REST is an interface that allows for the secure connection of two separate computer systems to exchange information. A client calling for REST APIs makes the server transfer relevant information to it. And that too in an interpretable format [10]. Both GraphQL & REST are fundamentally different. Here are a few of these differences:

1. GraphQL is a query language, while REST is an architectural concept for network-based software.
2. REST faces problems of over fetching (getting more than the required data) and under fetching (the opposite) - a limitation absent in GraphQL
3. GraphQL offers rapid product development with changes to be made only from the client side.
4. GraphQL supports schema stitching in a simple format – unlike its REST counterpart.
5. Unlike GraphQL, REST does not provide type-safety and document auto-generation.

2 Methods

2.1 Design of Datalogging Software

2.1.1 UML Design Steps

One of the most important technical implementations that has been done during this project work, is the development of a datalogging application. In short, this is a software module that exports the data from the “weather station”, to the “Dimension Four” cloud storage. Some aspects to consider with this application, is that it should handle Modbus and/or API connectivity to the weather station, be able to transform the data to a suitable format for the Dimension Four platform, and then export the data via API calls.

Due to using the free version of Dimension Four, the project is under a 100mb storage constraint that has to be taken into account; thus, the software should also be able to keep only the freshest records in the cloud storage, and have a "first in, last out" approach to deleting old records when inserting new data. For a more permanent storage of the data, one could create the option for an on-prem SQL-server export in the application, that can store more historic data as time goes on. In short, the following requirements for this datalogging application has been collected below:

Functional Requirements:

Set up connection through MODBUS and/or API with the weather system.

Do necessary data transformations to format data

Export data to Dimension Four platform

Handle/remove old data in Dimension Four platform

Some way of interacting with the software (either GUI or console commands)

Non-Functional Requirements:

Written in C#

Should run on Windows as OS, or Linux with Mono installed

After gathering the requirements, a basic use case diagram can be constructed. A tentative version was created, as seen below (Figure 2-1).

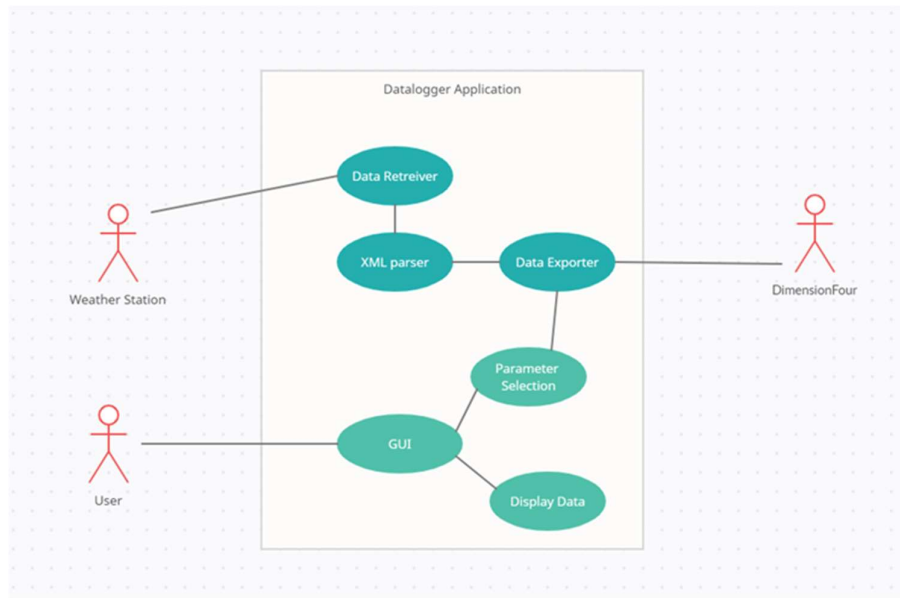


Figure 2-1: The class diagram of datalogging software

The main actors in this situation are the Weather Station, the DimensionFour tenet and the user interacting with the datalogging software. The use cases are interconnected

From the requirements and use case diagram, we can start to conceptualize hypothetical classes using a domain model (Figure 2-2) and figure out the necessary connections.

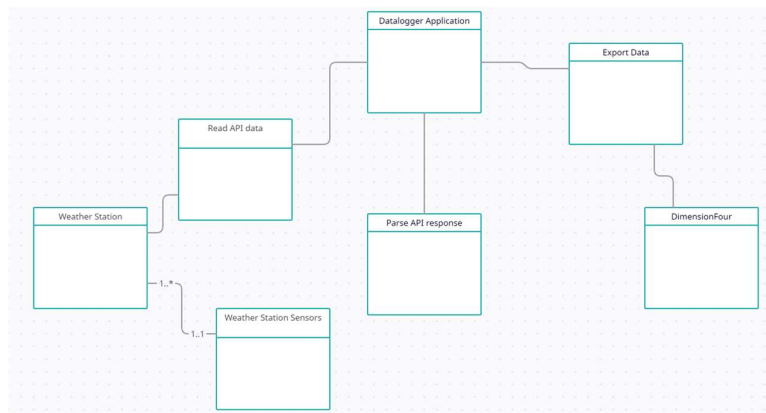


Figure 2-2: Domain Model of the datalogging software

This allows us to complete the first and initial class design (Figure 2-3). The below class diagram shows the final version of the application, built on the Proof-Of-Concept to retrieve data from the weather station, and export it to DimensionFour.

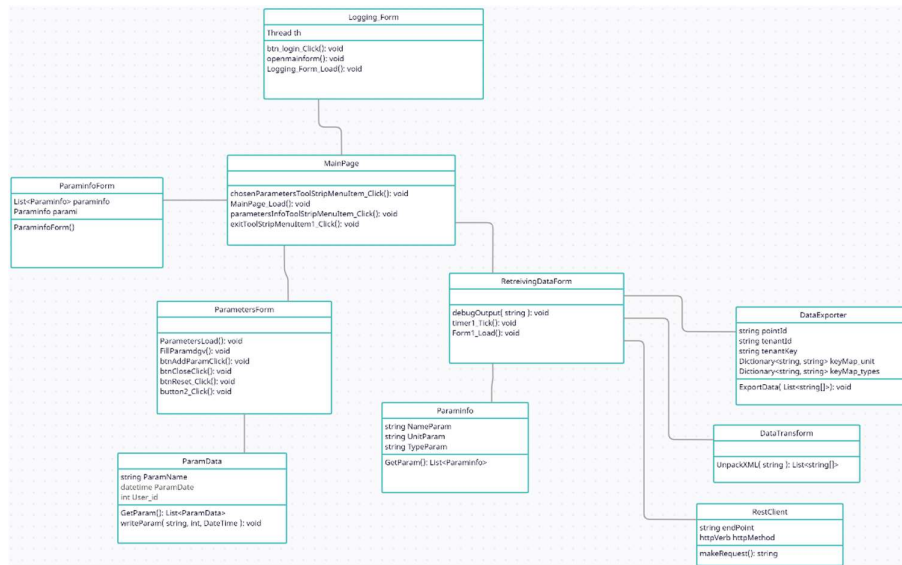


Figure 2-3: The final class diagram of the datalogging software

A few more GUI elements were added, and a dynamic query builder was made with mapping config files for units and types.

2.1.2 Implement and Testing

In order to develop the application, two software have been used. SQL Server and Visual Studio are these software. Defining a database in SQL Server and creating a windows application in Visual studio (#C), these are two structural parts of Datalogging Software.

2.1.2.1 Login Form

To create the security of this software, a login form with the desired user and password has been developed that the user can enter the software by entering her/his username and password. (Figure 2-4) For this purpose SQL Server software has been used.

The screenshot shows a Windows application window titled "Logging_Form". At the top center is the logo for "ISN Universitetet i Sørøst-Norge". Below the logo, the title "Login To Weather System" is displayed in a large, bold, blue font. Underneath the title, there are two input fields. The first field is preceded by a user icon and the second by a padlock icon. At the bottom of the form, there are two buttons: a blue "Login" button and a light blue "Exit" button.

Figure 2-4: Logging Form

2.1.2.2 Main Page

After entering the software, you can see the "Main Page" Form (Figure 2-5).

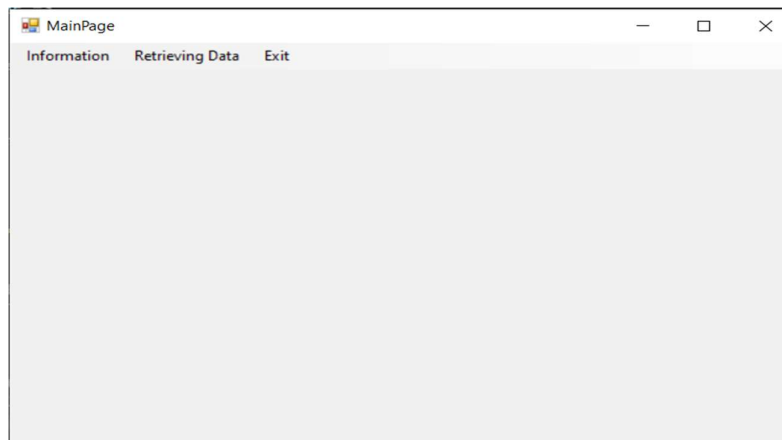
The screenshot shows a Windows application window titled "MainPage". It features a menu bar with three items: "Information", "Retrieving Data", and "Exit". The main area of the window is a large, empty light gray rectangle.

Figure 2-5: Main Page Form

In this page, by choosing the menu you can witness the following information:

- Parameters' information
- Chosen Parameters

2.1.2.3 Param Information

Parameters' information section you can find all parameters that are in the Weather System. A view of this part can be seen in Figure 2-7.



The screenshot shows a window titled "Param_info" with a table of parameters. The table has three columns: NameParam, UnitParam, and TypeParam. The first row is highlighted in blue.

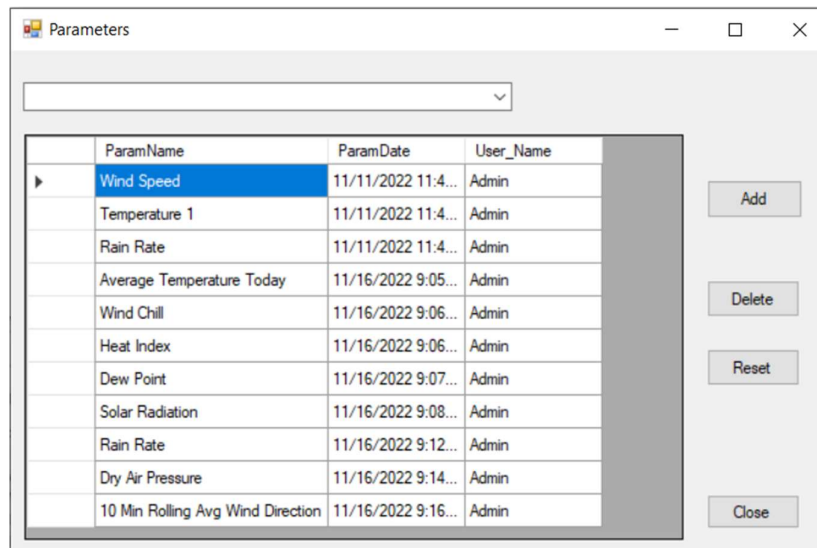
NameParam	UnitParam	TypeParam
mtTemp1	Celsius_Degree	Temperature
mtAdjBaromPress	hPa	Adjusted Barometric Pressure
mtRelHumidity	%	Relative Humidity
mtRainToday	mm	Rain Today
mtWindSpeed	m/s	Wind Speed
mtAdjWindDir	Degree	Adjusted Wind Direction
mtRawWindDir	Degree	Raw Wind Direction
mt3SecRollAvgWindSpeed	m/s	3Sec Roll Avg Wind Speed
mt3SecRollAvgWindDir	Degree	3Sec Roll Avg Wind Direction
mt2MinRollAvgWindSpeed	Degree	Min Rolling Avg Wind Direction
mt2MinRollAvgWindDir	Degree	Min Roll Avg Wind Direction
mt10MinRollAvgWindSpeed	m/s	Average Wind Speed
mt10MinRollAvgWindDir	Degree	Min Roll Avg Wind Direction
mt10MinWindGustDir	Degree	Min Wind Gust Direction
mt10MinWindGustSpeed	m/s	Min Wind Gust Speed

A "Close" button is located at the bottom right of the window.

Figure 2-6: Parameters information Form

2.1.2.4 Chosen Paramameters

Chosen Parameters (Figure 2-7) is an additional part of the project is for improvement of the project. Now all the data send to Dimension Four, but by using this part, this equipment will be given to the users to choose the favorite parameters to send to Dimension Four. The first part has been developed and the effect of this additional part to Dimension Four can be developed for upgrading the software for the future.



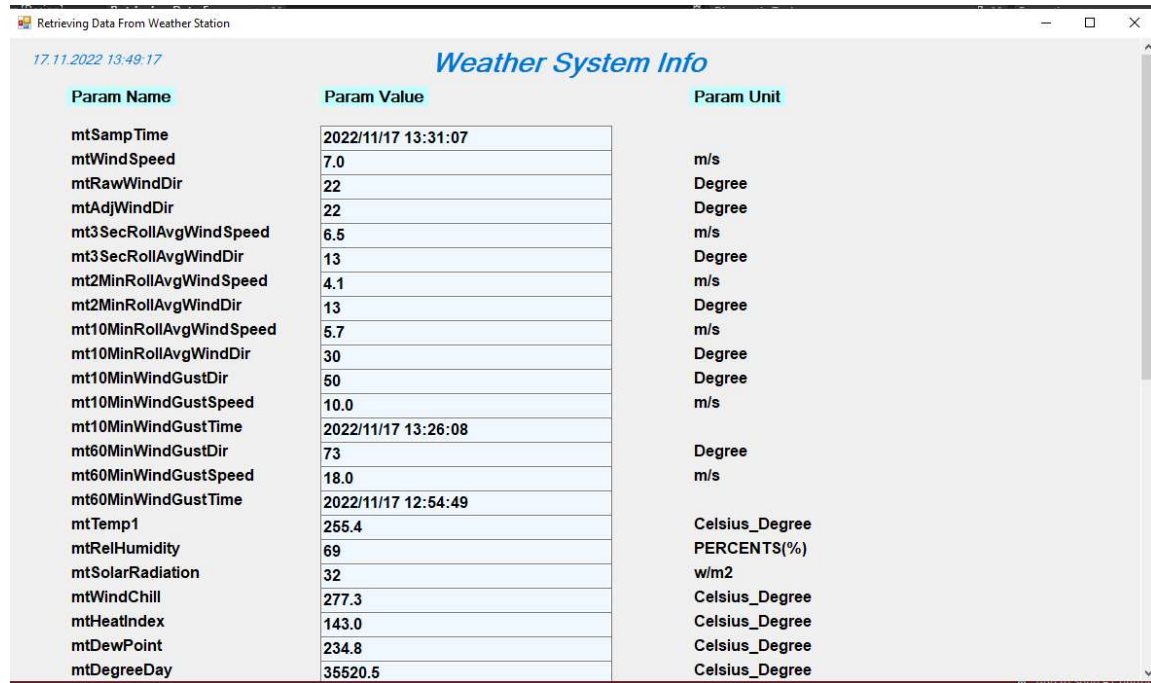
The screenshot shows a window titled "Parameters" with a table of chosen parameters. The table has four columns: ParamName, ParamDate, and User_Name. The first row is highlighted in blue. There are buttons for "Add", "Delete", "Reset", and "Close" on the right side of the window.

ParamName	ParamDate	User_Name
Wind Speed	11/11/2022 11:4...	Admin
Temperature 1	11/11/2022 11:4...	Admin
Rain Rate	11/11/2022 11:4...	Admin
Average Temperature Today	11/16/2022 9:05...	Admin
Wind Chill	11/16/2022 9:06...	Admin
Heat Index	11/16/2022 9:06...	Admin
Dew Point	11/16/2022 9:07...	Admin
Solar Radiation	11/16/2022 9:08...	Admin
Rain Rate	11/16/2022 9:12...	Admin
Dry Air Pressure	11/16/2022 9:14...	Admin
10 Min Rolling Avg Wind Direction	11/16/2022 9:16...	Admin

Figure 2-7: Chosen parameters Form

2.1.2.5 Retrieving data Form

The next menu belongs to main feature of the software, “Retrieving Data”. Based on the storage space in Dimension Four, the interval time of the retrieving data is every 10 minutes. The sample of retrieved data can be seen in Figure 2-8.



Param Name	Param Value	Param Unit
mtSampTime	2022/11/17 13:31:07	
mtWindSpeed	7.0	m/s
mtRawWindDir	22	Degree
mtAdjWindDir	22	Degree
mt3SecRollAvgWindSpeed	6.5	m/s
mt3SecRollAvgWindDir	13	Degree
mt2MinRollAvgWindSpeed	4.1	m/s
mt2MinRollAvgWindDir	13	Degree
mt10MinRollAvgWindSpeed	5.7	m/s
mt10MinRollAvgWindDir	30	Degree
mt10MinWindGustDir	50	Degree
mt10MinWindGustSpeed	10.0	m/s
mt10MinWindGustTime	2022/11/17 13:26:08	
mt60MinWindGustDir	73	Degree
mt60MinWindGustSpeed	18.0	m/s
mt60MinWindGustTime	2022/11/17 12:54:49	
mtTemp1	255.4	Celsius_Degree
mtRelHumidity	69	PERCENTS(%)
mtSolarRadiation	32	w/m2
mtWindChill	277.3	Celsius_Degree
mtHeatIndex	143.0	Celsius_Degree
mtDewPoint	234.8	Celsius_Degree
mtDegreeDay	35520.5	Celsius_Degree

Figure 2-8: The sample of retrieved data from weather System

This part of the software has been developed by creating a .NET REST API in C# using Visual Studio. [11]

In this part, a class that is called RestClient has been made, and it has some different parts as bellows:

- Create the Model
- Making a GET Request
- Making a POST Request

A list “responseValues” has been developed for retrieved data that stored the data in it. Then it used for showing data in a dynamic form with dynamic textboxes and labels to show the data for user.

In the next step the retrieved data has been sent to Dimension Four and will store there for monitoring the data.

2.1.2.6 API response parser

As the API response is returned in XML format, it was necessary to parse the data so that it could be indexed and handled dynamically by a List or an array. A separate class was made for

this purpose to isolate the logic and call upon the functionality when needed. The class `DataTransform` contains an `UnpackXML` function, that takes the API response as a string input, and then transforms it to an XML object using a .NET native `System.Xml` library. Once the response is converted to an XML object, the function loops over any XML nodes, and retrieves the parameter names and measurement values, and stores them in a string array in the format of ["parameterName", "parameterValue"]. These string arrays are then added to a List of string arrays, and thus all the XML nodes are collected in a single data list, where parameter names and their values can be accessed pair-wise.

2.1.2.7 Export to DimensionFour

One challenge of working with both REST API technology, and as we discovered with GraphQL, is to construct dynamic queries. While the use case is widely spread on the internet as a common feature implemented by many developers, there are no standardized way of doing it in existing libraries. Common libraries such as .NET GraphQL focus on building GraphQL services (retrieving queries, sending responses). The most suggested solution therefore, is to dynamically concatenate strings to construct a final query.

The first initial POC version of the datalogging application used a query of the following format in C#:

```
string query = @"{"query": "mutation CREATE_SIGNAL($timestamp:Timestamp!$pointId:ID!" +
fields.ToString() + @") {signal {create(input: {pointId:$pointId signals:[" + signalQuery.ToString() +
@"]}) {id timestamp createdAt pointId unit type data {numericValue rawValue} } } }",
  "variables": {
    "pointId": "" + pointId + @""",
    "timestamp": "" + DateTime.Now.ToString("yyyy-MM-ddTHH:mm:ss.ffffff") +
@"" " + variables.ToString() + @"
  }
}"

.Replace("%pointId%", pointId)
.Replace("%timestamp%", DateTime.Now.ToString("yyyy-MM-ddTHH:mm:ss.ffffff"));
```

The main challenge, is to place the parameter values correctly into the query, such as the sensor data and `pointId`, while also ensuring that the query adheres to the GraphQL syntax. If the syntax is incorrect by a single symbol, the query will not be accepted by the API. Some experimentation led to a correct format, which could then be used to build a dynamic query for the export.

There are several steps for implementing this. First, the input data is of the format `List<string[]>`, containing the pair-wise data from the weather station created by the API response parser. This List of string arrays is looped over, to access the parameter name and

value. These variables are then concatenated to constant strings to construct each GraphQL field individually. The API also requires a declaration of query variables, so a key-value list is constructed. Likewise, a declaration of Fields and the datatype before passing Field values is required. In short, the query syntax can be described by the following pseudo-code:

```
mutation CREATE_SIGNAL($timestamp: Timestamp! $pointId:ID!

                                $Field_1: String! ...
                                $Field_n: String!)

{signal{create(input:{pointId:$pointId
                                signals:[
                                    unit: Celsius_Degrees
                                    value: 25
                                    type: Average Temperature of the Day
                                    timestamp: 01/10/2022 12:00:00:000 ]})}}

variables: {
    pointId: 12345,
    timestamp: 01/10/2022 12:00:00:000,
    Field_1: 25,
    ...
    Field_n: 50
}}
```

The red highlights indicate roughly the parts that are dynamically added through string concatenation. The final feature added to this class, was a lookup for the measurement unit and type. These key-value pairs are defined in separate .json-files, and are then accessed when building the query fields to include unit and type for the measurements. The final concatenation construction can be seen below:

```
dataList.ForEach(data =>
{
    fields.Append($"${data[0]}:String!");
    signalQuery.Append(" {unit: " + keyMap_unit[data[0]] + " value:${data[0]} + " type:\\\" + keyMap_type[data[0]] + "\\\"
timestamp:$timestamp}, ");
    variables.Append($"\\n\\n{data[0]}\\n\\n{data[1]}\\n\\n");
});
```

And the query string is finally constructed by:

```
string query = @"{"query": "mutation CREATE_SIGNAL($timestamp:Timestamp!$pointId:ID!" + fields.ToString() +
@") {signal {create(input: {pointId:$pointId signals:[" + signalQuery.ToString() + @"]}) {id timestamp createdAt pointId unit type
data {numericValue rawValue} } } }",

    "variables": {

        "pointId": "" + pointId + @",",

        "timestamp": "" + DateTime.Now.ToString("yyyy-MM-ddTHH:mm:ss.ffffff") + @"" + variables.ToString() + @"

    }

}"

.Replace("%pointId%", pointId)

.Replace("%timestamp%", DateTime.Now.ToString("yyyy-MM-ddTHH:mm:ss.ffffff"));
```

Note that the C# string function .Replace() is used to insert pointId and Timestamp that is common for every query sent, and thus doesn't have to be retrieved and added dynamically.

2.1.2.8 GitHub

GitHub was used as code repository for this project, so that several people could work on the C# solution simultaneously. The repository changes were managed either by the console application or GitHub Desktop, according to personal preferences. The repository can be accessed at the following link:

https://github.com/MartinOse/Datalogging_Application

2.2 Development of Data Monitoring Application

This chapter presents the development of data monitoring web application that fetch and visualize stored data by the data logger application on Dimension4 GraphQL cloud server.

2.2.1 Web Technologies used for the Web Application

The data monitoring application is a single page web application (SPA) that developed by modern and popular technologies being used these days. This allows to deploy the application on a cloud service and access it from anywhere regardless of hardware and operating system. Thus, the application can be run on a mobile phone, tablet or an ARM processor computer. Moreover, there is no need to run the application in physical hardware in lab, home or office and instead, user just need to open a browser and visit the application URL. The application is deployed on Netlify cloud service and has an active development environment which means whenever the GitHub repository of the application is updated, it automatically will be deployed on Netlify, and the application URL shows latest update.

The data monitoring application developed in React JS library using JavaScript, CSS, and HTML. Logged data will be sent to a space, created on Dimension4 with the name of Capricorn 2000EX Weather Station. Sensor and Measurements points are created inside of this space to store signal data sent by the data logger application. Then on the main screen, some important data such as temperature, pressure, wind speed etc will be fetched from the space on Dimension4 GraphQL server. This is done by sending a post request including query to request data from the points created before.



Figure 2-9: Technologies are used for developing the data monitoring web application.

2.2.1.1 User Interface UI

-Initial user interface (UI) of data monitoring application is created by a raw HTML and CSS template that is taken from [4]. Then it is customized and integrated with React JS library. The data sent and stored on Dimension4 Cloud by the data logger application will be shown on the main screen and within different pages of the application. Presented data on the main screen,

is plotted in charts, and it is possible to access the charts by clicking on each of them to load its chart. The historical page is created to show the last 100 records stored on Dimension4. The data is presented on a table with id, type, raw value, and time. To generate a report from this data, a button is implemented and whenever the user clicks on the button, the report will be created as a PDF file will be downloaded which includes the latest data. The data on this table consists of 40 different measurement data stored on D4 from the weather station by the data logger application.

2.2.1.2 Data Monitoring Application Logic

The application consists of pages and components that have common header and navbar. The data after sending request to Dimension4 cloud and normalizing the data, it will be stored in a global state within the application. Storing the data in the global state will reduce the number of requests to just one request to Dimension4 cloud. Therefore, the data can be used in different components without extra request to Dimension4 cloud. The data will dynamically fetch every 10 minutes and store in the state. The User can change this time by going to historical data page and enter a new sampling time in minutes. Error handling for Dimension4 API URL is also implemented to show error message when API call to D4 server is not established and therefore data is not fetched to be shown on the main screen. Therefor instead, unable to fetch data from API will be shown in this case.

Regarding security of application, to avoid exposing the API key in front-end, the API key for D4 account is stored in environment variable which later will be set on a cloud host server. If the key was hard coded inside the application code, anyone could access the source code in web browser and use the API key to send data to D4 account.

3 Result

Figure 3-1 shows main screen of data monitoring application. As it can be seen in the figure, data is presented. The temperature line chart also visualized available data with day and time. By clicking on temperature chart, the app navigates to temperature chart route and show a bigger size of the chart. All parameters shown on screen, has a same feature and user is able to see detailed data on its specific chart (see Figure 3-2).



Figure 3-1: Data monitoring web application main screen.

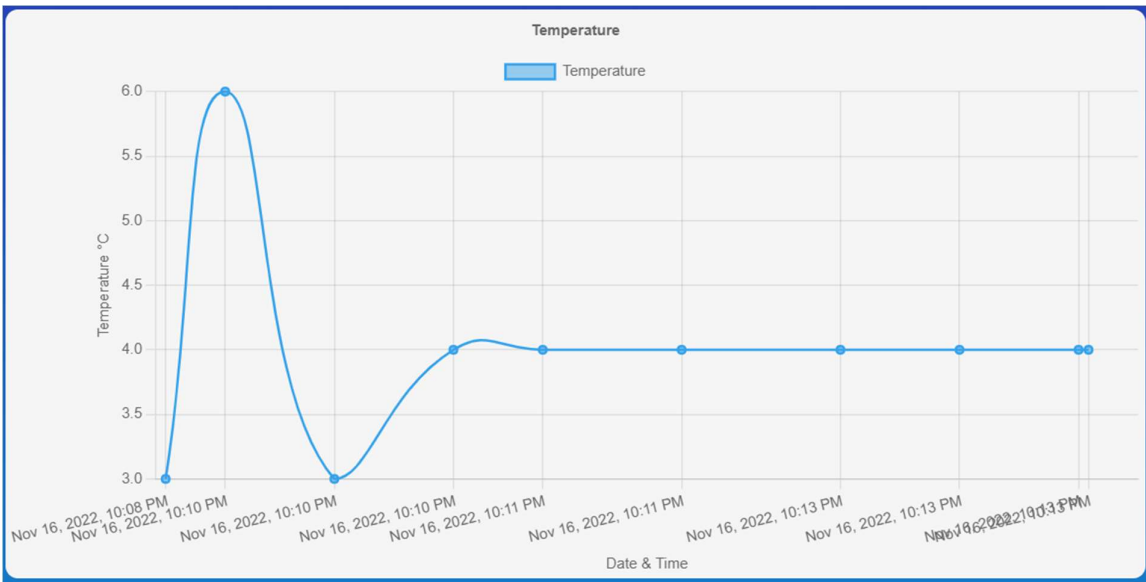
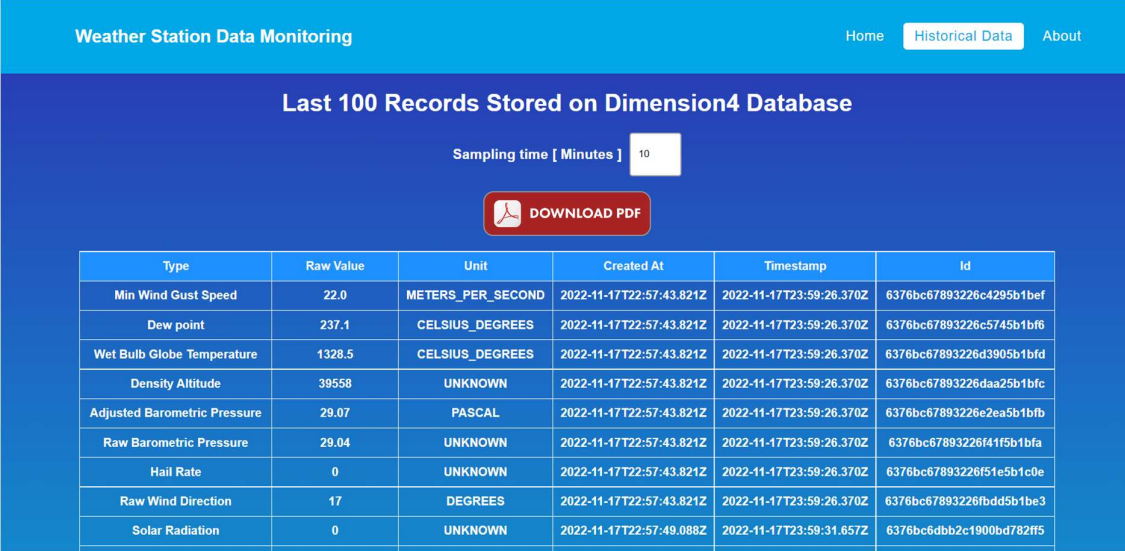


Figure 3-2: Full size of temperature chart with mode details.

By navigating to historical page, last 100 recorded stored will be shown on a table. Here user can download latest data in PDF format by clicking on DOWNLOAD PDF button.

Sampling time is also can be changed by entering a new value in minute (see Figure 3-3).



Weather Station Data Monitoring

Home Historical Data About

Last 100 Records Stored on Dimension4 Database

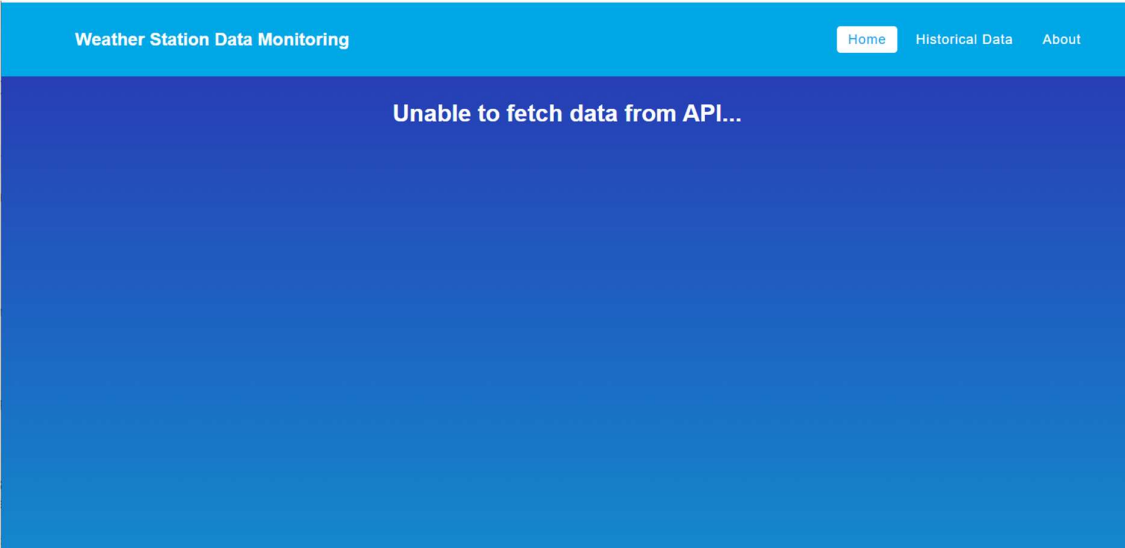
Sampling time [Minutes] 10

DOWNLOAD PDF

Type	Raw Value	Unit	Created At	Timestamp	Id
Min Wind Gust Speed	22.0	METERS_PER_SECOND	2022-11-17T22:57:43.821Z	2022-11-17T23:59:26.370Z	6376bc67893226c4295b1bef
Dew point	237.1	CELSIUS_DEGREES	2022-11-17T22:57:43.821Z	2022-11-17T23:59:26.370Z	6376bc67893226c5745b1bf6
Wet Bulb Globe Temperature	1328.5	CELSIUS_DEGREES	2022-11-17T22:57:43.821Z	2022-11-17T23:59:26.370Z	6376bc67893226d3905b1bfd
Density Altitude	39558	UNKNOWN	2022-11-17T22:57:43.821Z	2022-11-17T23:59:26.370Z	6376bc67893226daa25b1bfc
Adjusted Barometric Pressure	29.07	PASCAL	2022-11-17T22:57:43.821Z	2022-11-17T23:59:26.370Z	6376bc67893226e2ea5b1bfb
Raw Barometric Pressure	29.04	UNKNOWN	2022-11-17T22:57:43.821Z	2022-11-17T23:59:26.370Z	6376bc67893226f41f5b1bfa
Hail Rate	0	UNKNOWN	2022-11-17T22:57:43.821Z	2022-11-17T23:59:26.370Z	6376bc67893226f51e5b1c0e
Raw Wind Direction	17	DEGREES	2022-11-17T22:57:43.821Z	2022-11-17T23:59:26.370Z	6376bc67893226fbd5b1be3
Solar Radiation	0	UNKNOWN	2022-11-17T22:57:49.088Z	2022-11-17T23:59:31.657Z	6376bc6dbb2c1900bd782ff5

Figure 3-3: Historical data page.

Figure 3-4 shows an error message that was created on purpose by setting the API URL to a wrong address to test error handling functionality.



Weather Station Data Monitoring

Home Historical Data About

Unable to fetch data from API...

Figure 3-4: Error message when the API URL is misconfigured or the cloud server is down.

4 Discussion and Conclusion

The data logging and monitoring applications are developed successfully with acceptable features. The weather data stored on Dimestion4 GraphQL server are presented in the data monitoring application in different forms such as text, chart, table and PDF. The application has an active development environment and further development, and optimization is in processing.

The calculated number of requests to D4 per day per each application is 144 times if the sampling time is set to every 10 minutes. The free plan of D4 allows 100000 API calls for free. Thus, there is no need to change plan and pay for more API calls. Based on initial sampling configuration (10 minutes) and application logic that optimized and reduced the number of API calls to just one call to D4 server and expose the fetched data throughout the application pages via global state, the maximum number of API call is estimated to 4320 for the data monitoring application and if data logging application set the same, the total number of API calls is 8640 which is far below the limit (100000).

Selection of just one value which is latest was a challenge when you have many data point with same title and the latest value should be filtered out and display on main screen. This solved by filtering the normalized data array and then selecting latest value in the normalized array. Another approach could be selecting the timestamp and compare, however there is a chance to receive data with wrong timestamp and the logic may break. This problem could be prevented by fetching just the last 11 sensor data, however for the historical page the last 100 records were needed, and this means another API call is required to send a new call to D4. Moreover, in this way there is no limit to add or remove data and the table in historical data always shows last 100 records data sent by data logging application to D4 and data can be varied.

Future works on the data monitoring application are as following:

Add logic to detect sunny and cloudy days state and update image on main screen.

Add a control panel with login page to allow the user to add a new sensor or data point, space etc.

Add option to change tenant id and key for Dimension4 within control panel.

Implement chart component and feed data dynamically to create charts just based on given data in one place and not extra component for each chart.

Work on responsiveness of application style according to visitor's screen dimension.

Use YR as a second reference to compare measured data and then present.

Use YR weather forecasting data to present the weather situation over the next few days. This also could be implemented using statistics and machine learning based on stored data and live that is available in data monitoring application.

The Datalogging Application also has some potential for improvements, including:

1. Improved GUI and navigation in the GUI
2. More stable and optimized handling of the data. Currently, there seems to be issues with the data refresh, and a possible unverified memory leak that would need to be investigated further.
3. Although some of the sensors returned faulty values, it would probably still be necessary to implement conversion between imperial and metric units. This was not done in the current version, as some of the temperature values seemed unreasonable even after converting to Celsius degrees, and thus it was not part of the main focus while implementing the application.

References

- [1] Hans-Petter. Halvorsen, Weather System document, Porsgrunn: University of South-Eastern Norway, 2022.
- [2] Christian Francis Aanning, André Skare Berg, Ahmed Gurhan, Victor Amaechi Igbokwe, Jishnu Unnikannan Nair, Cuong Hong Nguyen
Design and Implementation of Weather System for Acquiring and Monitoring of Weather Data, Porsgrunn: Telemark University College, 2012.
- [3] Ahmed Gurhan, Design and Development of Windows Store Application for Measurements and Monitoring, Porsgrunn: Telemark University College, 2013.
- [4] Jon Keeping. Example of a responsive webpage making use of media queries for a weather app [Website]. Available: <https://github.com/JonUK/responsive-web-weather-app>
- [5] RedHat.com, 2019
available: <https://www.redhat.com/en/topics/api/what-is-graphql>
- [6] Fauna.com, 2021
available: <https://fauna.com/blog/graphql-mutations>
- [7] GraphQL.org
<https://graphql.org/learn/queries/>
- [8] Tutorialspoint.com
available: https://www.tutorialspoint.com/graphql/graphql_schema.htm
- [9] G2.com
available: <https://www.g2.com/products/graphql/competitors/alternatives>
- [10] Ronak Ganatra, GraphQL Vs. REST APIs, 2021
available: <https://hygraph.com/blog/graphql-vs-rest-apis>
- [11] Kindsonthegenius.com, 2021
available: <https://www.kindsonthegenius.com/how-to-create-rest-api-in-net-using-c-and-visual-studio/#t1>
- [12] Nakhuva B., & Champaneria, T., Study of various internet of things platforms, 2015
International Journal of Computer Science & Engineering Survey, 6(6), 61-74
- [13] Vandikas, K., & Tsiatsis, V., Performance evaluation of an IoT platform.
2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies (pp. 141-146). IEEE

Appendices

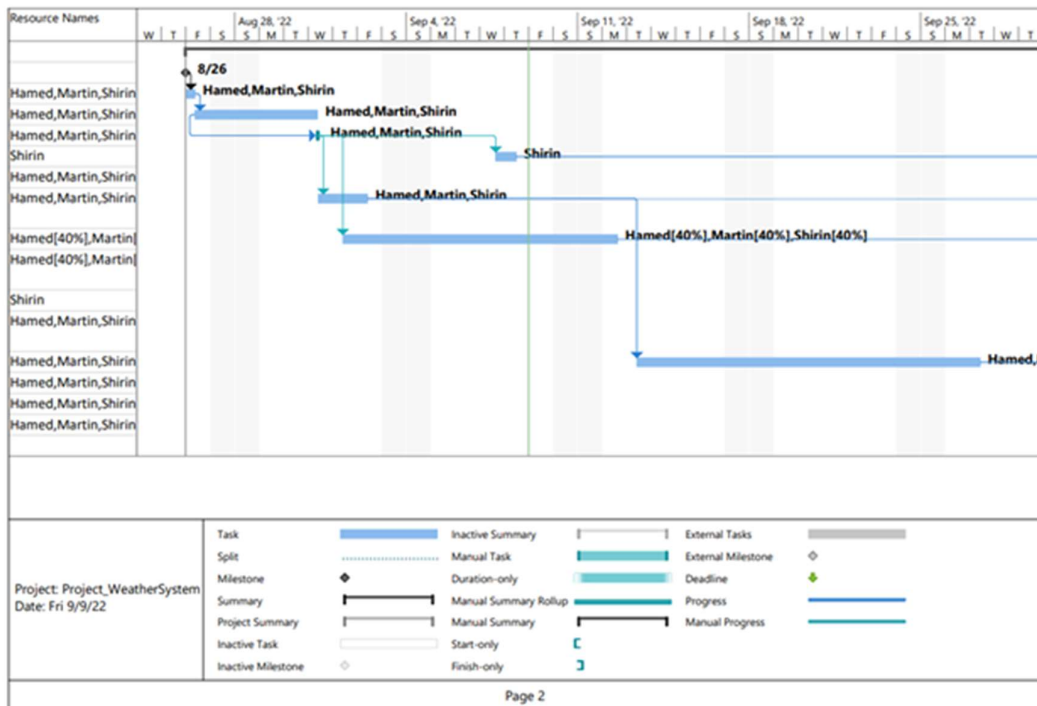
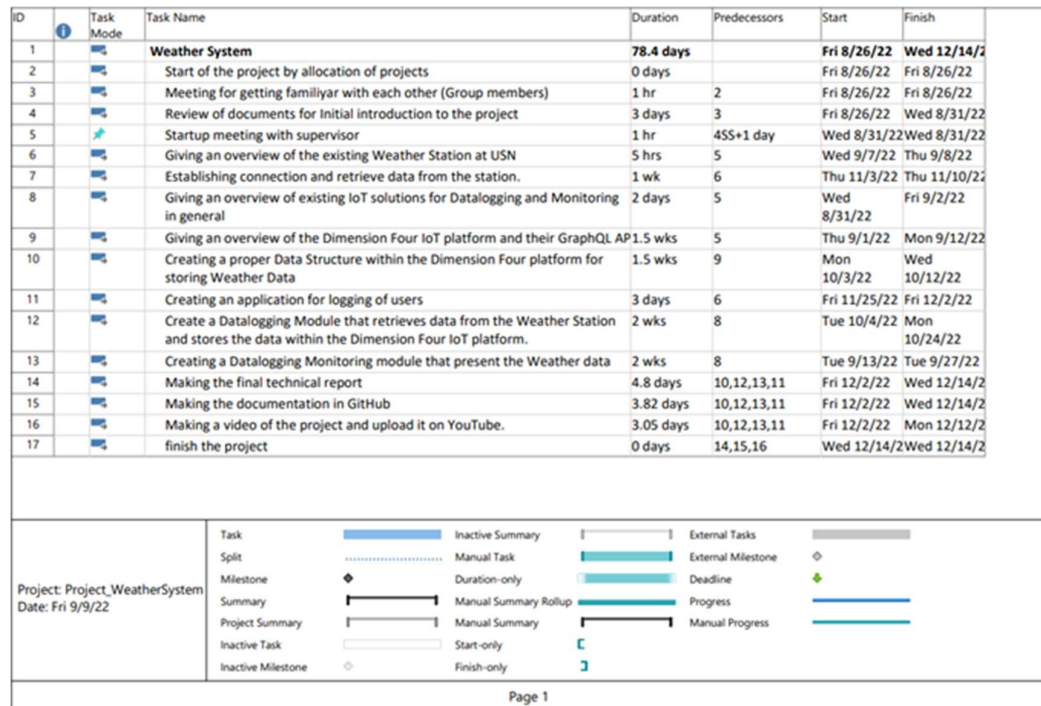
Appendix A <Project description>

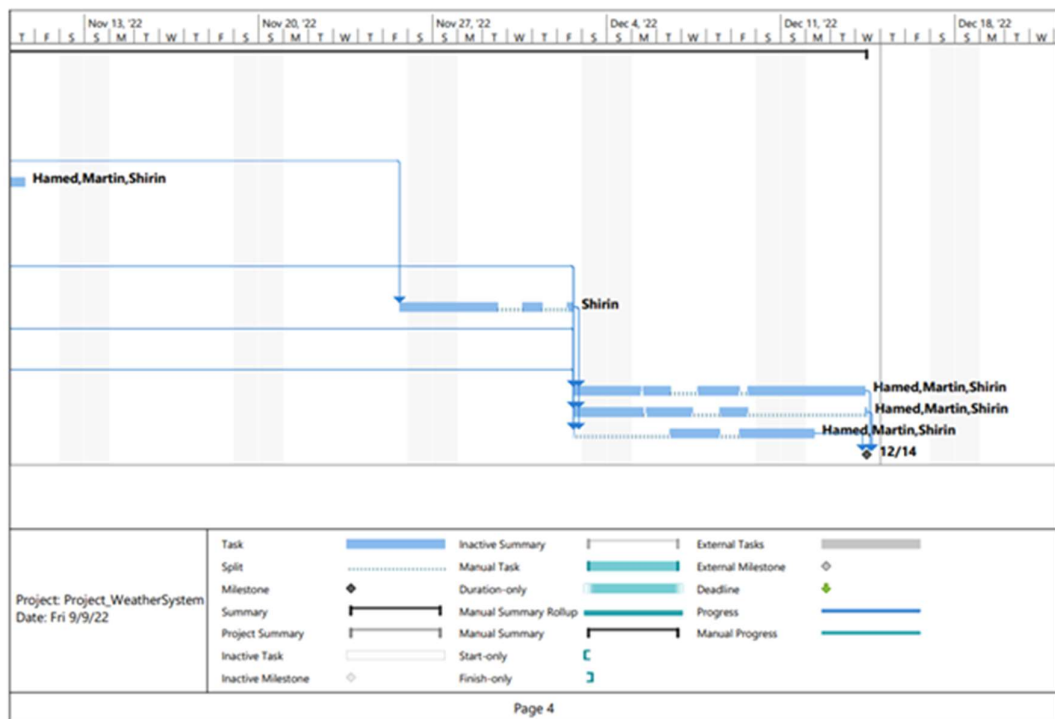
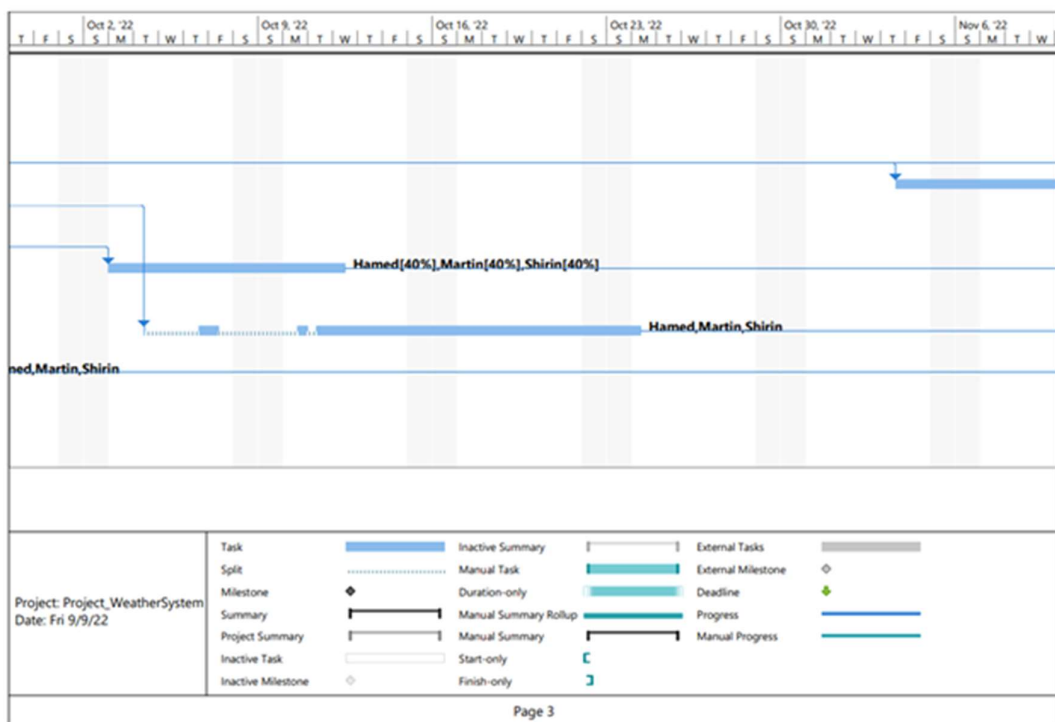
This project is trying to get the weather status from the Weather Station at USN and log the data from it through a Weather Microserver and send data to Dimension 4 and monitor the data which is given. For this purpose, three software programs are developed, retrieving the data from Weather Station, sending the data to Dimension 4 and monitoring the data.

.....

Appendix B <The Gant Chart>

The Gantt chart of the project can be seen as follows:





Appendix C

GitHub repository for the datalogging application:

https://github.com/MartinOse/Datalogging_Application

Weather Data Monitoring application access URL:

<https://usnp.netlify.app>

Remark: If new appendix ➔ New page (i.e. don't put several appendices on the same page)